Role Delegation as Multi-Agent Oriented Dynamic Composition

Denis Jouvin ; Salima Hassas

LISI – Université Claude-Bernard Lyon I,
bât. Nautibus, 43 boulevard du 11 novembre 1918,
69622 Villeurbanne, France
djouvin@lisi.univ-lyon1.fr

Abstract. In the context of highly heterogeneous, complex and dynamic systems, we can compare distributed object based and component based approaches, that were first centered on the idea of *composition* and *modularity*, where *dynamicity* has become an issue afterward, with MAS approaches, whose first concern was the *dynamicity*, and where *composition* was in the beginning rather considered as an implementation issue. Now, both communities tend to foster the gap. Our work is part of this trend, by exploring the notion of *situated composition*, where each agent or component can have its own compositional perception of the rest of the system, and by considering that an agent is actually an extended component. We achieve agent dynamic composition in terms of delegation and agent proxies. This way, component aggregation accounts for synchronization and existing interaction patterns. Agent conversations are the natural canvas supporting compositional relationships, and agents are recursively decomposable, without having to introduce other organizational constructs that might disturb agents fundamental properties.

Introduction

In today's highly heterogeneous, complex, and distributed systems stand the inescapable issues of system *interoperability*, and, close to the latter, *composability* — that we define as the ability by a system to be decomposed in several components, or integrated as a component into a larger system.

Composability is of course a key idea in the software engineering history, and has lead several people to center software architecture paradigm on a global composition "unit" model, referred to as *module*, *object*, *holon*, *component* or whatever the name is, in order to cut the system down into pieces, reduce each piece's complexity, and increase reusability. A system is composed of smaller sub-system which are connected, or interact together in a well defined way, which is a natural way of dealing with complexity. We are right on the way to the idea that "all is object".

The more interoperable the composition "units" (objects, components) are, the easier it is to reach profitable system-wide properties like the ability to evolve, scalability, maintainability, and some form of robustness.

However, while in the past system *composability* was mainly considered at design-time, as an architectural, modeling and implementation issue, it is now more and more becoming a runtime issue.

In the Multi-Agent System (MAS for short) community, interoperability is a major concern, but the dynamic aspect was also essential from the beginning. However, MAS *composability* is still a slippery concept, for different reasons.

In this paper, we try to foster the gap between both approaches, with respect to our problematic, by first bringing together the *component* and *agent* concepts, and by defining a composition paradigm, in the Multi-Agents context, by Multi-Agent means.

We first explore dynamic composition in the MAS literature, and also in component-based models, and (distributed) object oriented models. Then, we refine the notion of composition by introducing *situated composition*. We propose a modeling tool to implement this paradigm based on a generalized dynamic delegation mechanism. Finally, we illustrate this approach by applying our model to the auction scenario, where auctioneers are considered as temporary components of a wider construct.

1 Scope, problematic and key issues

1.1 Agents, Objects and Components: Complementing Approaches

This section aims at showing that Distributed Objects or Components Oriented and Agents Oriented approaches are actually complementing approaches, sharing some common goal, especially interoperability, but having chronologically addressed the same problems at different times.

From a systemic point of view, interoperability is both a system-wide property, and a system internal property. In other words, the difference between the inter- and intra-level of a system is, most of the time, only a matter of *granularity*. Thus, the ability to compose or decompose the system is a very important aspect of interoperability. It has always been the first concern in modern software engineering approaches, like *modular* or *object oriented* design. It is still the major concern in more recent Distributed Object models, such as OMG CORBA [1], and Component Based models, like Enterprise Java Beans [2] or OLE/COM and .NET [4] architectures.

However, today's other key issue is the dynamicity. In order to modify a system at runtime, which is more and more needed, the coordination, not only the static relationships, between components should be known, and if possible explicit[1]. Component Based systems are a step toward addressing this issue, as well as distributed object models, as we see in section 2.

Multi-Agent Systems, on the other hand, represent probably the approach most focused on its dynamic aspect. Agents in a MAS are highly autonomous entities, which are as much defined by their interaction patterns as they are by their internal (static) architecture. In MAS context, interoperability is increased thanks to this autonomy, which reduces agents dependencies, and by the interaction pattern standardization

[1] Throughout this paper, we define as explicit any information which is expressible and accessible by the system itself, to some extent. Implicit information is not represented and inherent to the system, observable only from outside. Explicit information is often meta-data.

and/or adaptation at different abstraction levels, depending on the type of MAS involved.

In the MAS community, however, the *composability* issue is most of the time secondary, and treated differently at the intra-agent level and at the inter-agent level. In other words, at these two different levels, the connection or interoperation between subsystems is considered of a different nature — which leads to the difficult problem of knowing where the limit between these two levels should be, in terms of complexity, in a given system. Thus, the "all is agent" proposition does not stand currently.

Inside agents, many researchers propose to use component based architectures (see section 2.1 for further details). Between and outside agents, either the question is not raised because the systems do not involve enough agents to motivate it; or some other organizational constructs are introduced; or people consider that organization emerges by itself dynamically, and thus do not necessarily follow any fixed compositional structure.

1.2 Targeted Multi-Agents Systems

In our context, we need to restrict the scope and the type of MAS concerned by our problematic. Our actual concern is the ACL[2] based MAS, especially the communication and interaction level, excluding both the cognitive aspect of complex agents, which takes place at a higher level, and other forms of communication, not ACL based, encountered in reactive MAS (like environment based communication for example). Why such restriction, apart from the fact that MAS is a vast domain?

First, because interactions are the essence of MAS, and this level of interaction is the most likely to be standardized, which is an important condition to interoperability — keeping in mind that "standard" does not necessarily mean *rigid*, *fixed* or *non-flexible*, but should be taken here in the *agreement* meaning of the term. Environment based communication systems, on the other hand, are communication paradigm which do not really use the concept of explicit interlocutor, thus composition takes a different turn and is not in our scope.

The second reason is that dynamic composition is a general enough issue to concern almost all systems, at this level, but too complex and specific at higher level. Our interest is the common part, the common behaviors, which we can factorize to maximize interoperability.

1.3 Key Issues

Our idea is that, even if organizational structures are by essence dynamic and possibly highly ephemeral in MAS, they necessarily follow well-defined decomposition patterns. The problem is to find out what such patterns should look like, and above all what such patterns have in common, in MAS, and with respect to other approaches.

Considering this, our work has been driven by those inter-related questions, grouped by the following issues:

---

[2] Agent Communication Language, defining Communicative Acts and Interaction Protocols

- **The unification between agents and component:** is it desirable to distinguish the composition intra- and inter-agents? Is it desirable to compose agents of agents, i.e. to be able to see a group of agents as one agent? Then, if so, can we say that an agent is an extended component, as we would say that a component is an extended object — *i.e.* will components be "agentified"?
- **The redefinition of the composition notion as Situated Composition:** How to redefine composition relationships between components/agents, if a global (hierarchical) composition structure is no more useful or adequate? How to integrate multiple decomposition structures or constructs?
- **The use of Delegation and Agent Proxies as a dynamic composition construct:** How to achieve such situated composition dynamically, while still keeping intact common MAS properties? What mechanism and interaction pattern can be used for this purpose?

After a more detailed analysis of existing models in section 2, we address these issues and questions in section 3.

2 Dynamic Composition in Existing Models

2.1 Dynamicity in Distributed Objects & Component Based Architectures

The Distributed Object community, largely represented by OMG CORBA, and the Component Oriented Programming community, are also two worlds getting closer as each one's problematic enlarges and overlaps with the other one.

Initially centered on remote invocation and distribution, it is obvious that infrastructures like CORBA [1] are more and more integrating the possibility to introduce a new object and object's class dynamically into the system, or replace an existing one, with minimal impact on the running system. A CORBA object is indeed not "hard-wired" into a system, since it is to be invoked remotely. In a sense, distribution forces the decomposition to have most of the dynamic composition requirements, including object *late-binding*.

In CORBA for example, such *late binding* is made possible by the Naming Service, or indirectly through other dynamic services such as the Relationship Service, Trading Object Service or Query Service. However, the most significant and detailed advance in term of system dynamic composition is the CORBA Component Model specification, which definitively build a bridge with Sun's Enterprise Java Beans (EJB) technology [2]. Objects are "componentified", and encapsulate their own dynamic "pluggability" with the running system, to some extent.

Symmetrically, Sun's EJB specification defines clearly how one can use CORBA as an underlying architecture and remote method invocation mechanism to fulfill it's own requirements. The purpose of this paper is of course not to go into those technologies details, however we need to emphasize the major similarities with Agent Oriented Programming approaches:

- Components are dynamically pluggable in their hosting platform environment, as agents do in an agent platform;

- They are self-registering, sometimes self-presenting, and have a well-defined and explicit life cycle, as agents;
- They usually offer asynchronous messaging to communicate, although they do not follow explicit protocol like agents — however, event registration models are a step toward explicitly defined interaction patterns;
- They are sometimes mobile, and message are forwarded accordingly, as agents;
- They can somehow impersonate several roles simultaneously (called interfaces or facets), although the notion of role is here merely an interface or list of methods, events and attributes, whereas agent roles (at the communication level) describe interaction rules and responsibilities.

Additionally, a component can itself be a component *container*, and thus publish other components without relying on dedicated services. It can be composed of, or reference other components at runtime, permitting dynamic composition. This property is not so obvious in MAS.

Why object and component systems can afford it, and not MAS? Because, object and component are weaker concepts than agent, in terms of constraints, and do not explicitly specify interactions patterns. It is the responsibility of the designer to check the global system behavior, but the complexity is still there. UML, for example, provides tools to model such interaction patterns, but the target system do not explicit nor normalize them so that they be perceived by components themselves, as meta-data.

MAS approaches go further on this aspect, as we discuss in section 3, and thus the "connection" between agent is not as trivial as it seems with components or objects.

2.2 Composability in Multi-Agent Systems

In the following section we analyze how composition paradigms are currently used in Multi-Agent Systems, at the intra-agent level first, and at the inter-agent level then.

Intra-Agent Components. Building MAS is definitely not an easy task. Agent are often complex entities, compared to mere objects or even components, and they can span a broad range of abstraction levels. The first approach used to deal with this complexity has been to use common software engineering techniques, especially component oriented approaches, in order to define reusable common components to facilitate each agent's design and implementation.

We have reviewed several agent development platforms relying on component reusing techniques. Among them are the platform Zeus [5], Bond [6], FIPA-OS [7], CASA [8], MADKit [9] and, to some extent, Jade [10] and RAJA [11].

One of the most complex aspect in agents is the communication overhead, especially the conversation[3] follow-up. As a consequence, MAS platforms designers have integrated *conversation handlers* components in their frameworks. Most agent platform and development environments, such as Zeus [5] and FIPA-OS[7], among others, propose such components as central building blocks. Finite State Machine (FST) engine components based systems, like Bond [6], have a similar mechanism, if we

[3] A conversation is seen here as an Interaction Protocol instance or occurrence.

consider only external events, i.e. messages, as transition trigger (internal events represent the coupling between FST components and the agent's kernel).

What is interesting to remark is that the communicational behavior of such components is very similar to that of an agent, as we explain in [13]. In this paper we argue that making conversation handlers agents on their own, while keeping a very simple default conversation handling behavior inside agents, is an important step toward satisfying the "all agents" idea, and has a lot of advantages, in terms of interoperability and dynamicity. "All is agent" does not mean that everything is to be viewed as an agent, of course, as there is a critical complexity and requirement level which is higher than in the case of object or even components. However, it does mean that whenever an entity requires or support the properties of an agent, it must be one of them in order to increase interoperability.

At this stage, we have already answered to the questions related to the unification between a component and an agent: these two concepts, or abstraction levels, share common grounds, and thus we believe that the supporting implementations and models, should share common mechanisms to increase the overall interoperability. For example, as both entail asynchronous messaging, they should share the *same* message transport mechanism, rather than duplicating it, in the agent and in the inter-agent communication layer, as it is most of the case in the above mentioned platforms.

An agent is an extended component because it satisfies component requirements. Agent and Component platforms should reflect this property as much as possible.

Symmetrically, designing an agent that would in the end be implemented as a mere component is not impossible. As we mentioned before, interaction patterns are already considered in Object Oriented modeling tools and languages, like UML. This type of regression would be comparable to designing by contract, when programming in an object oriented way, using a procedural programming language.

Agent Groups. Most of MAS advantages are linked to properties defined at the inter-agent level (thus the word *multi* in MAS). Beside temporary organizational structures inherent to any conversation, some people have introduced other organizational constructs, with a longer life-time, for example the agent groups or channels in MadKIT [10], or the agent hives in Hive [12].

These structures are separate from agents, they are not seen as normal agents; thus we cannot really talk about recursive decomposition. Actually, they define fixed structural levels in the system, the agent level, and the "group" level, with very specific functionalities. As a result, the designer cannot handle complexity in a very scalable way. In other words, if a given system's complexity increase in time, and one agent is no more adequate to handle a too complex function, we can not easily split that agent into several agents.

This problem is actually similar to that seen in the intra-agent composition. As soon as decomposition units (components) are not equivalent, or at least compatible at different levels, we are faced with fixed abstraction or granularity levels and we end up with a not so scalable solution. That's why the unification or at least compatibility between component and agent, and between agent and groups of agent (agent composability) is, to our opinion, a necessity.

3 Composing by Delegation

3.1 Conversations as Basic Organizational Construct

We briefly introduced the notion of *conversation* in sub-section 2.2. While very vague in the early MAS literature, conversation has become a key to understand MAS communication layers. This notion is at the center of our approach.

We define a conversation as a succession of messages (communicative acts) exchange between several agents, not necessarily two, following a well-defined interaction pattern called Interaction Protocol (IP). Interaction protocols can be seen as the common reference knowledge between agents at the communicational level. They can also span many abstraction levels. What we here refer to as communicational level is the *de facto* level of complexity that is ruled by IP so far in the MAS community.

An IP defines several Agent Roles (AR), to not confuse with the role concept in UML, which defines the set of responsibilities of one interlocutor participating to the conversation. Several roles may be impersonated by one unique agent. Agent Communication Languages specifications give a very good definition of the Interaction Protocols, their use, and how they should be defined.

Conversations are the instantiation of interaction protocols in actual systems, and are by themselves a basic organizational construct, in that they define a relationship between interlocutors, and their *roles*. This will help us to define the concept of Situated Composition in the next section.

3.1 Situated Composition

The annoying part in the intuitive definition of composition/decomposition is not only that it is *hierarchical*, but the fact that it is *globally hierarchical*.

We can say that, in Distributed Object systems and Component Based systems, this definition does not hold, and the notion of composition has evolved toward a less restrictive idea: multiple objects can share common components, and we are talking about object graphs rather than object trees when we describe object oriented systems. Components are connected almost without restrictions other than compatibility (the only purely hierarchy is when components or objects are *private*, meaning that they are only visible to their enclosing *parent*).

Static composition is of course more likely to accept a global hierarchy, because it corresponds usually to one synthetic viewpoint, that of the designer. In a dynamic, highly complex and distributed environment, this is no more acceptable. actually, even in a static context, we often have several parallel views of the system, involving different decompositions articulated around these viewpoints, as modern design tools like UML emphasize.

Having said this, how to define *composition* differently, to fit the nature of our environment? Our idea is to define it as *situated composition*. *Situated* because each agent/component has it's own view of other collaborating components/agents. Several, possibly hierarchical or not, decomposition can coexist, because they all are relative to the viewpoint represented by one or more agent or component.

Situated composition is not equivalent to "one piece composed of many pieces", but rather to "see many pieces as one, from one given viewpoint". Besides, the former composition notion can be seen as a special case where sub-components are private.

Now that we have generalized the composition concept, relieving the decomposition uniqueness constraint, remains the problem of finding an adequate construct which still provide the dynamicity related advantages.

Our proposition is to situate further the concept of component with respect to an *agent role*, instantiated in a given *conversation*. As a direct consequence, when the conversation is finished, the composition, that is to say, the situated relationship, terminates. At higher level, some relationship may remain, either in enclosing conversations, or in the behavior or memory of agents. At the communicational level, though, a component or agent's connections to the system are nothing more than the conversations they are involved in. The *conversations* become the canvas supporting dynamic composition graphs.

This view of situated composition fully complies with the concept of agent acquaintances found in the MAS literature. However, stating that agents have acquaintances that can change dynamically only states that there is no restriction or constraint in the way agent are "acquainted" with each-other, but this does not provide any standard way of grouping or simplifying complex structures from a given viewpoint, which is what composition / decomposition is supposed to provide. Nor does it define any standard behavior common to all agents to perceive and modify the system compositional structure dynamically. These problems are left to higher abstraction levels, in other words, in the designer's hands. Agent acquaintances, at the communicational level (in the scope of our study), is merely equivalent to agent references. In current MAS architectures, the facilitator or matchmaker entities actually impersonate a great part of the compositional behavior of the system.

In the next subsection, we provide a simple organizational pattern and construct which does not contradict any property required by agent acquaintances, but defines a standard way to modify acquaintance compositional structures, with minimal impact on the running system, while accounting for the synchronization of the conversations. The system structure is then partly defined by the conversations delegation graph.

3.2 Dynamic Delegation and Proxy Agents: Making Things Explicit

Our idea is that there is a need to make this modeling construct (situated composition) explicit to agents / components, not only the designer, and to factorize and standardize all related common behavior at the communicational level, through a well defined dynamic delegation act and the related semantic.

One reason is that giving access to compositional meta-data to agents will allow them to manage or at least have influence on organization dynamically, which is, again, necessary in MAS. Thus, we can "agentify" the compositional behavior of the system more easily, and, above all, *in a standard and interoperable way*.

The common behavior that we need to factorize is the dynamic establishment of a compositional relationship. We can classify corresponding actions as:

– Add/Remove a component to the system, and make it available/unavailable to other components;

– Replace an existing situated component with a new one;
– Group several components into one, or split one component into many;

The first class of action is already fully achievable with MAS platform. However, the second and third are more specific. We argue that a generalized delegation mechanism, based on the definition of proxy agents in conversations, is a generic enough construct to allow these classes of dynamic composition action, at the communicational level at least.

Definition of Dynamic Delegation. An *Agent Proxy* is in our context an agent that will impersonate one or more roles in a conversation, on behalf of another agent. This definition does not prevent the proxy to still have interactions with the "proxied" agent, that we will refer to as the originator, in order to fulfill his new agent role.

Dynamic Delegation is the act of introducing at "conversation-time" such a proxy, usually at the beginning of the conversation, but not necessarily. We give a more formal definition of dynamic delegation in section 4, which explains how such mechanism is performed, along with the model which goes with it.

For reference, our definition of delegation differs from and significantly extends the limited KQML *brokering* communicative act, which does not allow any role substitution for a conversation involving more than two participants, as for the FIPA ACL *proxy* communicative act. We discuss these limitations in [13]. These communicative acts are interesting as they express the strong need for delegation in MAS, however they are too restrictive by not allowing correct delegation of n-participants conversations, to be used as organizational constructs.

What is of interest about an agent proxy, as we define it, is not the fact that it is an indirection level, but the fact that delegation is a way to define how agents can, when needed, add an indirection level in a complex communicational environment, and in a standard and synchronized way, because this is a task agents are likely to perform quite often in future flexible and dynamic systems. Besides, simple indirection implies the idea of a mono-channel forwarded communication, whereas delegation is a more complex notion which implicates the notion of agent role in an n-interlocutors environment, synchronization, responsibility, security (although the two latter aspects are out of scope of this document).

Additional Motivations. As we state in [13], there are other motivations to standardize such generalized delegation behavior among agents. The first is to allow easy Interaction Protocol adaptation through predefined agent proxies. We explain how such mechanism represents a scalable and interoperable infrastructure in MAS, enlarging the notion of protocol compatibility.

We also show how proxies can handle a part of the protocol and conversation complexity, by defining generic protocols. In such case, the facilitator agents or facilitation infrastructure can take a full benefit of it by indexing or understanding indirect client-provider mappings in its knowledge base.

4 Proposition and Application

4.1 Conversation Parameter

Before going further, we need to introduce a *conversation object*, or *conversation parameter*. This object is a communicational meta-knowledge, perceivable and accessible by agents, partly representing a conversation instance. It is composed of :

1. an identifier for the conversation, if not already included in the message parameter;
2. the name of the associated IP, if not already present;
3. the identity of each role, represented by {role name, agent names} key-pairs ;
4. if it exists, the identity of the proxy, or the complete proxy chain, for any given role — so that we can keep explicit track of the original role, for security reasons;

This conversation object is meant to be a complement to classical message parameters (namely, :reply-to and :sender in FIPA messages), and also to propagate agent identities in the case of complex multi-participant interaction protocols. Accessible during the conversation, for example in message headers, it eliminates any ambiguity raised by the *:reply-to* parameter which would not be enough in our case.

This parameter's semantics is not at the same level as FIPA envelop parameter. The conversation object describes the role attributions, and intermediate proxies, in terms of communicational responsibilities, while the envelop information is used solely for message delivery and tracing.

4.2 Dynamic Delegation Interaction Protocol

The meaning of dynamic delegation action is to substitute an agent during a whole conversation possibly involving more than two participants, and of an arbitrary complexity. This action is itself described here in term of message exchange in figure 1.

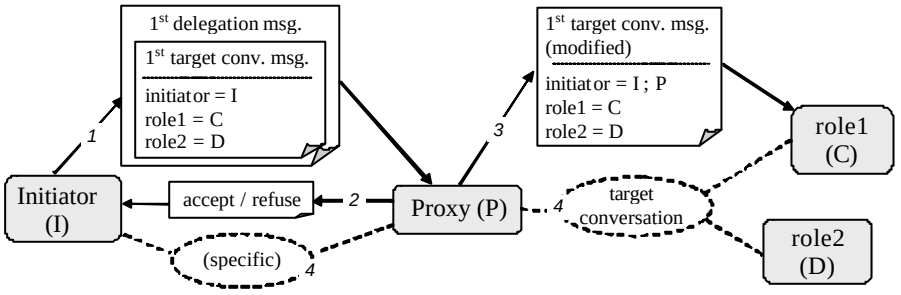


Fig. 1. Dynamic Delegation Interaction Protocol

On figure 1, we can see that the first delegation request encapsulates the target conversation first message. We have detailed the conversation parameter in this message. This parameter stipulates which agent impersonates which agent role.

After accepting the delegation, the proxy impersonates the role of the requestor, on behalf of the requestor, with respect to other participant, for the rest of the conversa-

tion. This is represented by the modification of the conversation parameter by the proxy (the line with "initiator = I ; P"), meaning that the initiator role, in the target conversation, is no more I's duty but P's duty, on behalf of I.

The proxy and requestor may simultaneously perform a private conversation, typically compatible with the original agent role we are substituting, where the proxy represents all other interlocutors, but not necessarily.

If the requestor is also initiator of the embedded conversation, which is the case on figure 1, there is no synchronization problem. Otherwise, the platform will forward any message received after the delegation request, to its proxy, if the latter accepts to proceed, of course. Actually, the way this dynamic redirection is effectively done depends on what kind of routing, naming and addressing the agent platform supports. But it would be based on the conversation parameter.

Important Properties. This mechanism has mainly three advantages with respect to dynamic composition:

- it propagates and keeps track of the requestor's identity and access right;
- it is non-destructive of the initial conversation's interaction pattern and protocol;
- it is very simple and involves only one (additional) message exchange.

What such delegation does is actually an agent substitution in the context of a conversation. Our idea is that this simple behavior should be supported by agent platform communication layer because it allows dynamic composition over the natural organizational canvas represented by agent conversations.

4.3 Implementation Scenario

The auction scenario described in [13], and prototype associated with it, was not initially intended to illustrate *composability* through dynamic delegation, but rather protocol adaptation. If we consider auctioneers as several components grouped into one by the auction proxy, though, we can state that "proxying-able" agents are ready for dynamic composition. Figure 2 briefly represents such auction.

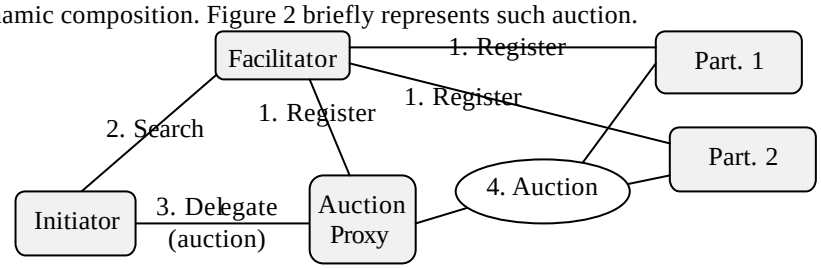


Fig. 2. Basic Delegation Pattern Applied to Auction

The lines and circles represent conversations, and the numbers represent their chronology. This pattern involves a facilitator, which represents the "agentification" of the system flexible structure. It can be considered internal or external to our auction sys-

tem, depending on how complex the system is, and how specific the facilitator's behavior should be.

However, in this scenario it is not easy to see where and how dynamic composition takes place, because auction is already a naturally distributed and service driven scenario where the auctioneer, let say a seller agent, explicitly search for participants.

Now, suppose an environment change such that we need to federate some participants — for example, in order to optimize the auction because some participants are far away. Figure 3 shows such federated auction. "D" lines represent the delegation conversation, "A" lines represent the auctioneer role in an auction, and "P" lines the participant role in an auction.

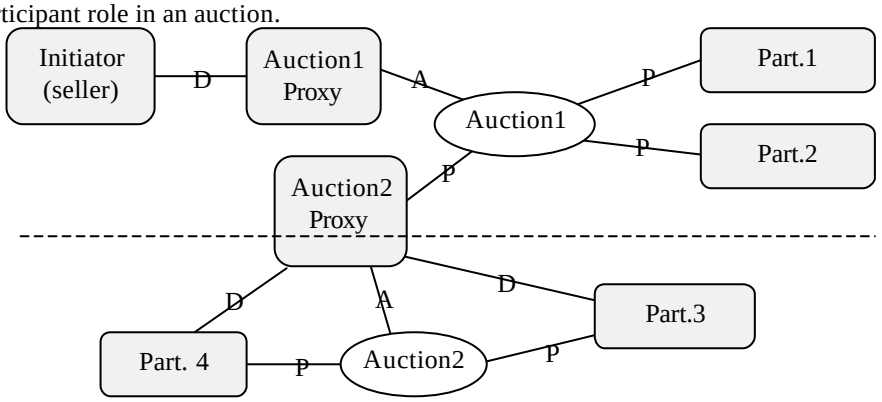


Fig. 3. Auction Federated by Agent Proxy

What is interesting is the way the second auction is dynamically set up: the initial conversation is not disturbed, and from the point of view of the initiator, participants 3 and 4 are still participating to the conversation. The second proxy is a new dynamic compositional node in the system.

This mechanism takes into account, and is not destructive with respect to, the existing synchronization and interaction pattern. Moreover, the initial identities of agents in the conversation remain intact, from the point of view of the initiator. In other words, agents not directly concerned will not have their compositional connections (acquaintances) with other components disturbed by the new structural change.

In other classical approaches, this change in the system structure would typically require to "replace" remote participants with intermediate agents used to federate the auction in some agents acquaintances. This would imply several changes either in the local facilitator behavior, and in the way the initial auctioneer search for participants.

We can of course recreate a comparable mechanism with component based approaches, but it would involved some specific communicational behaviors, like redirection, conversation object maintenance and interpretation. Our proposition is to integrate and factorize this composability and interoperability related behavior in the platform, and to design MAS accordingly.

Conclusion

In this paper we have defined the notion of *situated compositional* as an extension to the intuitive and classical composition notion, and we have defined a composition paradigm built on the agent conversation, by using *dynamic delegation*. We have also presented a simple auction scenario to illustrate our approach.

The rational behind this is to unify both concepts of agent and component: an agent being an extended component, an "agentified component", in a similar way as components are extended objects. Both approaches have indeed common motivations, and share common grounds and requirements at the communicational level, which should be factorized and standardized in agent based and component based development and runtime platforms.

We show that Dynamic Delegation is a modeling tool that provides interoperability and dynamic composition at the communicational level of MAS, considering conversations as a structural canvas.

The perspectives of this work are first to identify and analyze other comparative scenarios involving system and system context change, in order to illustrate further the potential use of this type of delegation, and to define or modify existing agent oriented design method so that they use this model.

References

1. The Common Object Request Broker: Architecture and Specification, CORBA 3.0 New Component Chapters, http://www.omg.org
2. Enterprise JavaBeans Specification, Version 2.1, 2002, http://java.sun.com/products/ejb/
3. Agent Management Specification, Foundation for Intelligent Physical Agent, 2001, http://www.fipa.org/
4. .NET Framework SDK Documentation, Microsoft, http://www.microsoft.com
5. Nwana, H., Ndumu, D., Lee, L., Collis, J.: ZEUS: a Toolkit and Approach for Building Distributed Multi-Agent Systems. ACM International Conference on Autonomous Agents (AA) 1999, Seattle, USA (1999)
6. Bölöni, L., Marinescu, D.: A Component Agent Model – from Theory to Implementation. TR, Computer Science Dept., Purdue University, West Lafayette, In, USA (1999).
7. Poslad, S. , Buckle, P., Hadingham, R.: The FIPA-OS agent platform: Open Source for Open Standards. PAAM 2000, Manchester, UK (2000)
8. Sesseler, R.: Building Agents for Service Provisioning out of Components. AA 2001, Montréal, Quebec, Canada (2001)
9. Gutknecht, O., Ferber, J., Michel, F.: Integrating Tools and Infrastructures for Generic Multi-Agent Systems. AA 2001, Montréal, Quebec, Canada (2001)
10. Bellifemine, F., Poggi, A., Rimassa, G.: JADE: a FIPA2000 Compliant Agent Development Environment. AA 2001, Montréal, Quebec, Canada (2001)
11. Ding, Y., Malaka, R., Kray, C., Schillo, M.: RAJA - A Resource-Adaptive Java Agent Infrastructure. AA 2001, Montréal, Quebec, Canada (2001)
12. Minar, N., Gray, M., Roup, O., Krikorian, R., Maes, P.: Hive: Distributed Agents for Networking Things. ASA/MA (1999)
13. Jouvin, D.: Utilisation de la délégation pour l'adaptation de protocoles de conversations entre agents. Journées Francophones d'Intelligence Artificielle Distribuée et Systèmes Multi-Agents (JFIADSMA), St-Jean-la-Vêtre, France (2000)